Lesson 2

Python structures and Loop

Previously in Python Base...

Variables;

Data types;

Operators;

Conditionals.

Introduction

Loops:

For, while

Python structures:

Lists, sets and tuple.

Grouping data

Indexing

Mutability

Loops

Loops

for and while

```
for X in Y:
    Do something
while Z:
    Do something
```

X can be an iterator in the range of a number Y or an item contained in Y, for example.
Z can be a condition: a > b or a variable.

Loops

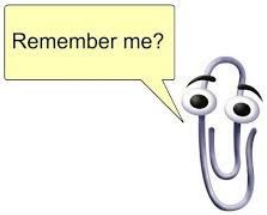
Examples:

```
for word in list_of_words:
    print(word)
```

```
for iterator in range(0,5,1):
    print(iterator)
```

```
a = 4
while a:
    a -= 1
```

```
while a > 0:
    a -= 1
```



range(start,end,step) or range(end)
In the latter, start and step are implicit, and when not declared step is 1 and start is 0.

Exercise

Given a string, split it and count how many elements there are in it.

```
string = “This could be any string you wanted, but now this is my sentence.”
string = string.split(“ ”)

word_counter = 0
for word in string:
        word_counter += 1
        print(word)
print(word_counter)
```

More of python structures

Lists, tuples, sets

Lists: defined between brackets “[ ]”, mutable, ordered (indexed)

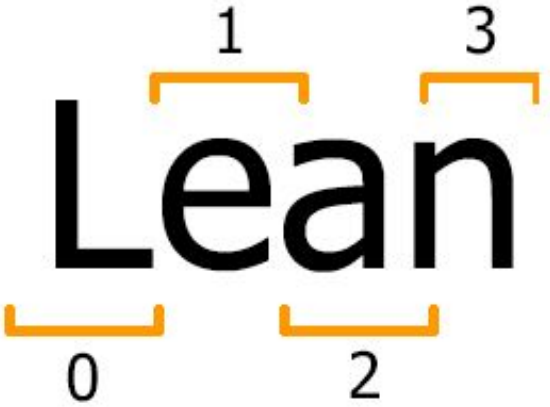
Tuples: defined between parentheses “( )”, immutable, ordered (indexed)

Sets: represented between braces “{ }”, mutable, unordered, unique values

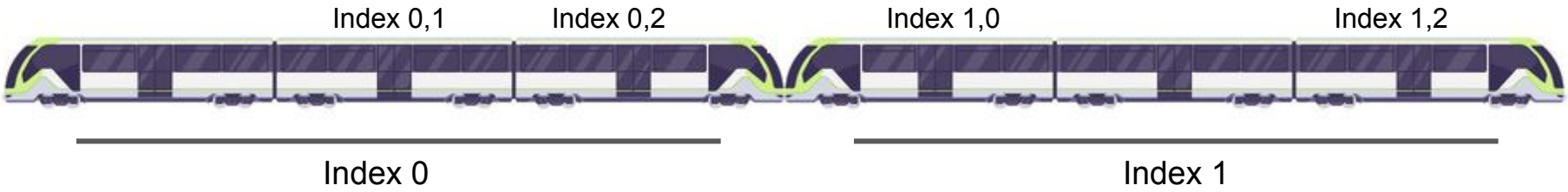
What does mutable/immutable and ordered/unordered mean?

Indexing

A string is a sequence composed by individual characters. Each character can be accessed by its position inside the string.



Indexing


Index 0,1
Index 0,2
Index 1,0
Index 1,2
Index 0
Index 1


Ferrovie.info

Indexing

shopping_list = “beer, chips, frozen pizza, water, instant noodles, ice cream”

How do we access individual values?

How can we count how many items are in our shopping list?

Indexing

shopping_list.split(“,”)

shopping_list = [“beer”, “chips”, “frozen pizza”, “water”, “instant noodles”, “ice cream”]

Before splitting: 60 elements
After splitting: 6 elements

Now we can access each individual item in our list by calling its index

The Python function len() returns the length of the iterable. I.e. how many elements are in it.

Exercise

Given a string, split it and get rid of the comma.

string = “This could be any string you wanted, but now this is my sentence.”
string = string.split(“ ”)

Exercise

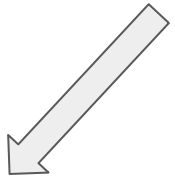
```
word_counter = 0
for i in range(len(string)):
        word_counter += 1
        if "," in string[i]:
                string[i] = string[i].split(",")[0]
print(word_counter)
```

Exercise

Given a string, split it and get rid of the comma.

```
string = “This could be any string you wanted, but now this is my sentence.”
string = string.split(“ ”)

word_counter = 0
for word in string:
        word_counter += 1
        if "," in word:
                string[string.index(word)] = word.split(",")[0]
print(word_counter)
```



Mutable vs Immutable

Python is dynamic. The variables can “change type” freely since there is no memory restriction. As we saw, even sequences (strings) can “change” their data type freely.

However, the same does not necessarily applies to every data structure in Python.

Groups of data may have a fixed number of elements or not. They can be composed exclusively of only one data type or more.

variable1 = “part 1”

variable1.append(“ part 2”)

variable1 will then be read as “part 1 part 2”.
What had 6 characters now has 13.

Mutable vs Immutable

We can also extend lists, for example:

```
list1                          =  [1,2,3,4]
list2                          =  [5,6,7,8]
list1.extend(list2)
```

```
print(list1)
[1,2,3,4,5,6,7,8]
```

list1 had 4 elements before the extend method and 8 elements after it. The number of elements changed.

```
list1.pop(-1)
```

And now it has 7 elements, the last one was removed.

```
print(list1)
[1,2,3,4,5,6,7]
```

Mutable vs Immutable

Specific items can be changed as well:

```
list1 = [1,2,3,4]
list1[0] = 6
list1[-1] = 6
print(list1)
```

[6,2,3,6]

Mutable vs Immutable

Specific items can be changed as well:

```
list1 = [1,2,3,4]
list1[0] = 6
list1[-1] = 6
print(list1)
```

```
[6,2,3,6]
```

This is what we just did in the previous exercise! We assigned a new value to an item in our list of words.

Exercise - 1

1. Declare 2 lists and print them.
2. Declare a 2 element tuple, where the first element is the first list and the second element is the second list and print it. *e.g. my_tuple = (list_a, list_b)*
3. Try to change the last element of the tuple (*e.g. my_tuple[x] = 79*) and re-assign the first element of the tuples' first element to any number.
4. Print the tuple and the first list.

Retry the previous operations skipping the first part of step 3

Exercise - 1

```
list_a = ["This is a number","The previous item is a lie","The cake is a lie"]
list_b = [42,43,44]
print(list_a, list_b)

my_tuple = (list_a, list_b)
print(my_tuple)

try:
    my_tuple[-1] = 79
    my_tuple[0][0] = 45
    print(list_a)
    print(my_tuple)
except Exception as error:
    print("There was a problem! {}".format(error))
```

```
['This is a number', 'The previous item is a lie', 'The cake is a lie'] [42, 43, 44]
(['This is a number', 'The previous item is a lie', 'The cake is a lie'], [42, 43, 44])
There was a problem! 'tuple' object does not support item assignment
```

Exercise - 1

```
list_a = ["This is a number","The previous item is a lie","The cake is a lie"]
list_b = [42,43,44]
print(list_a, list_b)

my_tuple = (list_a, list_b)
print(my_tuple)

try:
    # my_tuple[-1] = 79
    my_tuple[0][0] = 45
    print(list_a)
    print(my_tuple)
except Exception as error:
    print("There was a problem! {}".format(error))
```

```
['This is a number', 'The previous item is a lie', 'The cake is a lie'] [42, 43, 44]
(['This is a number', 'The previous item is a lie', 'The cake is a lie'], [42, 43, 44])
[45, 'The previous item is a lie', 'The cake is a lie']
([45, 'The previous item is a lie', 'The cake is a lie'], [42, 43, 44])
```

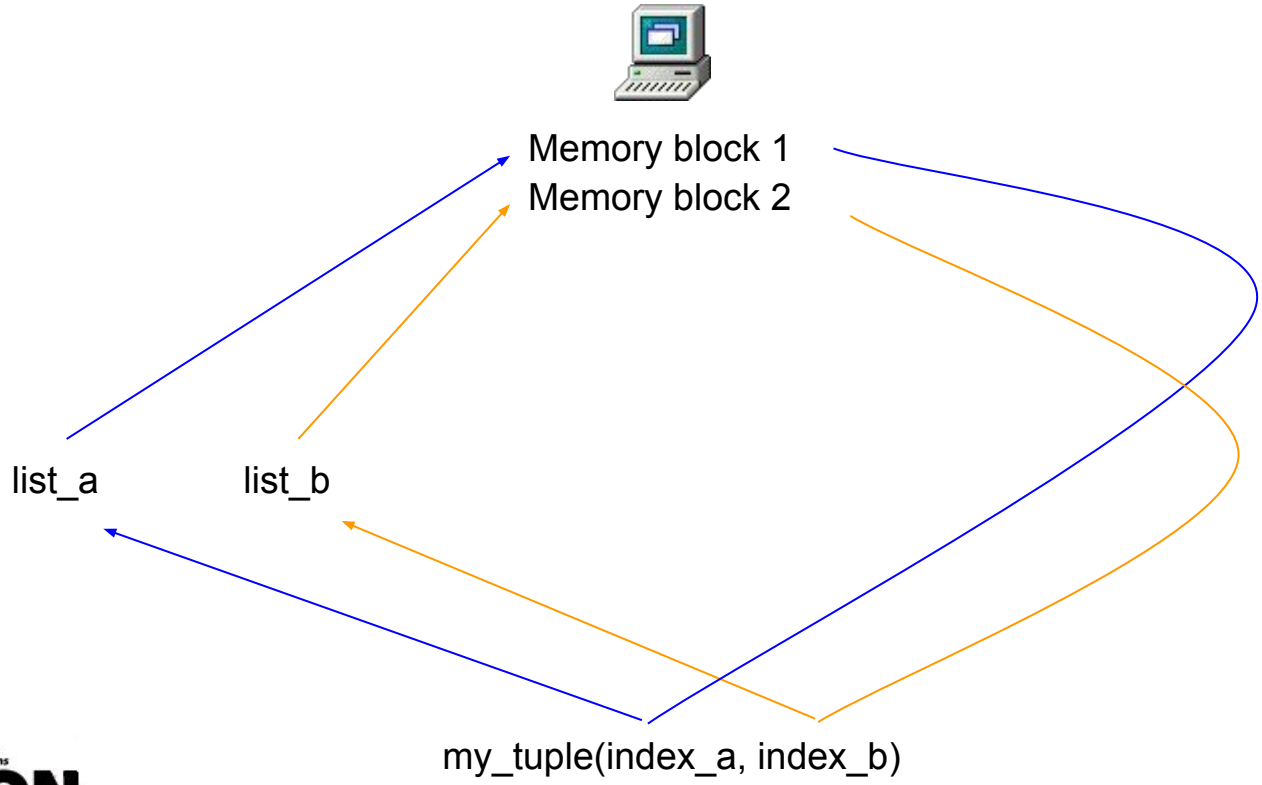
Exercise - 1

We notice two things:

1st - We cannot re-assign an element of a tuple. It is immutable. However, we can re-assign elements within mutable elements that compose a tuple.

2nd - Why did our first list changed if we didn’t even touch it?

Exercise - 1


Memory block 1
Memory block 2
list_a
list_b
my_tuple(index_a, index_b)

Exercise - 1

```
list_a = ["This is a number","The previous item is a lie","The cake is a lie"]
list_b = [42,43,44]
print(list_a, list_b)

my_tuple = (list_a[:], list_b[:])
print(my_tuple)

try:
   # my_tuple[-1] = 79
    my_tuple[0][0] = 45
    print(list_a)
    print(my_tuple)
except Exception as error:
    print("There was a problem! {}".format(error))
```

```
['This is a number', 'The previous item is a lie', 'The cake is a lie'] [42, 43, 44]
(['This is a number', 'The previous item is a lie', 'The cake is a lie'], [42, 43, 44])
['This is a number', 'The previous item is a lie', 'The cake is a lie']
([45, 'The previous item is a lie', 'The cake is a lie'], [42, 43, 44])
```

Exercise - 2a

Create a list that represents the output of a mathematical function.

1. Perform the mathematical operation (5 ** (-i*0.01)) 100 times with step of 1 and assign each output to a successive element of a list.

Extra: To visualize the plot, you can import matplotlib library. To install it just type "pip install matplotlib" in your console window. At the beginning of your python script, type: "from matplotlib import pyplot" **This will be covered into more detail in the upcoming lessons.**

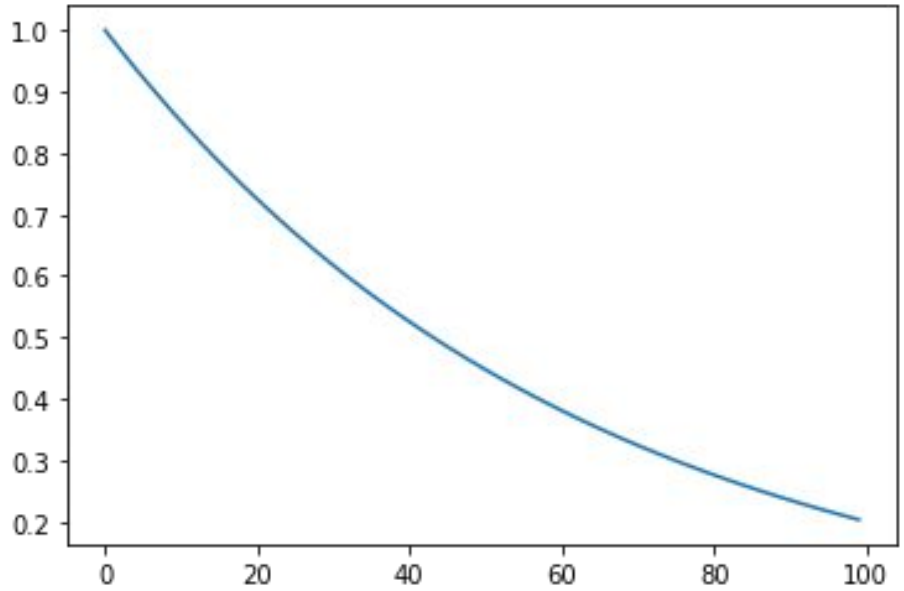
Then type in a new line "pyplot.plot(my_list)" and then in another line pyplot.show()

Exercise - 2a

```
from matplotlib import pyplot
my_list = []

for i in range(0,100,1):
    output = 5 ** (-i*0.01)
    my_list.append(output)

pyplot.plot(my_list)
pyplot.show()
```



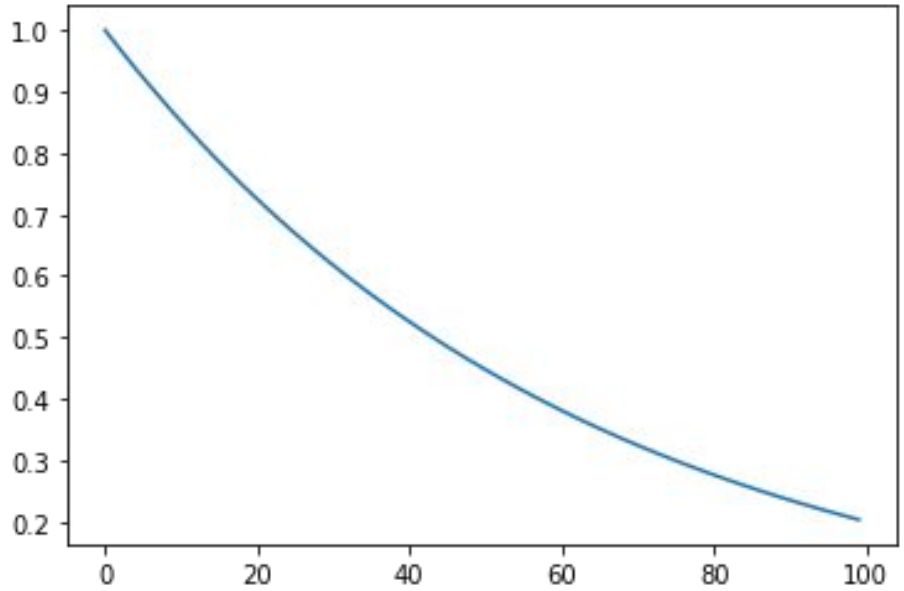
Retry this exercise using the WHILE loop instead of FOR

Exercise - 2b

```
my_list = []


i = 0
while i < 100:
    output = 5 ** (-i*0.01)
    my_list.append(output)
    i += 1

pyplot.plot(my_list)
pyplot.show()
```



2D (nested) - Lists

Lists

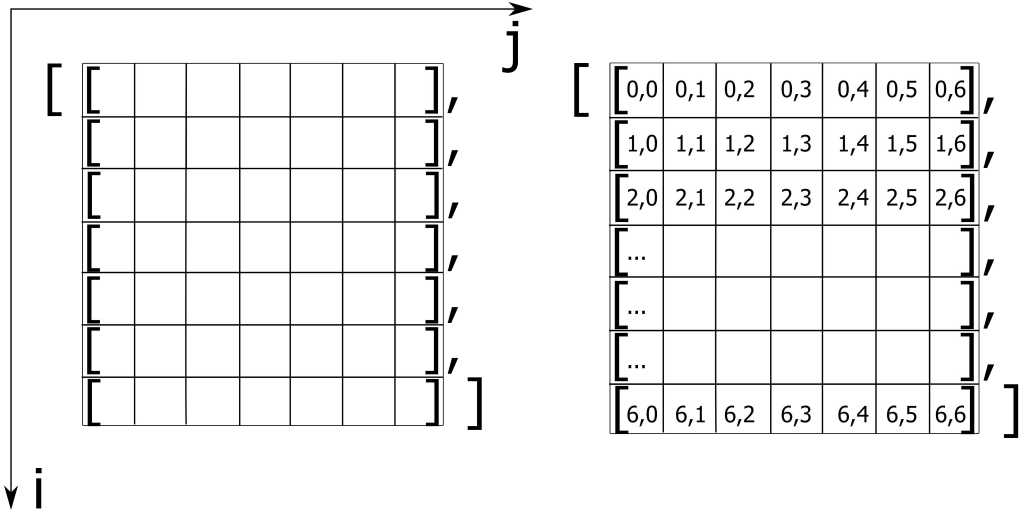
2D lists:

```
my_list = [[1,2,2,1],[2,1,1,2],[1,2,2,1]]
print(my_list[1])
print(my_list[1][0])
```

```
[2,1,1,2]
2
```

Looping through 2D lists:

```
my_list = [[1,2,2,1],[2,1,1,2],[1,2,2,1]]
for i in range(len(my_list)):
    for j in range(len(my_list[i])):
        print(i,j,my_list[i][j])
```



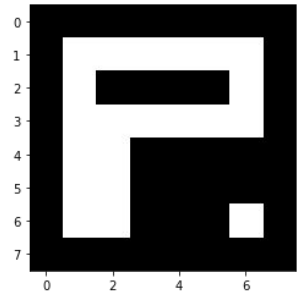
Lists

Now let us focus on something more useful: images.

```
import cv2, os
my_image = os.getcwd()+"\\image.png"

image = cv2.imread(my_image,-1)
print(image)
```

```
[[  0   0   0   0   0   0   0   0]
 [  0 255 255 255 255 255 255   0]
 [  0 255   0   0   0   0 255   0]
 [  0 255 255 255 255 255 255   0]
 [  0 255 255   0   0   0   0   0]
 [  0 255 255   0   0   0   0   0]
 [  0 255 255   0   0   0 255   0]
 [  0   0   0   0   0   0   0   0]]
```



Now you can clearly see that the image we opened is in fact a list of lists. To be more precise, it is a list with 8 elements, where each element has 8 items. These are our x and y coordinates.
This is a monochromatic image, the values are either 0 or 255.

Lists

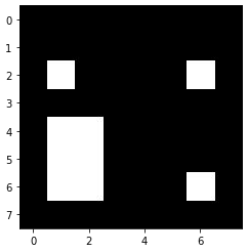
```
import cv2, os
my_image = os.getcwd()+"\\image.png"

image = cv2.imread(my_image,-1)

image[1] = [0,0,0,0,0,0,0,0]
image[3] = [0,0,0,0,0,0,0,0]

print(image)
```

```
[[  0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0]
 [  0 255   0   0   0   0 255   0]
 [  0   0   0   0   0   0   0   0]
 [  0 255 255   0   0   0   0   0]
 [  0 255 255   0   0   0   0   0]
 [  0 255 255   0   0   0 255   0]
 [  0   0   0   0   0   0   0   0]]
```



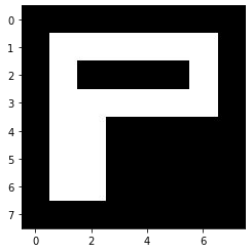
```
import cv2, os
from matplotlib import pyplot as plt
my_image = os.getcwd()+"\\image.png"

image = cv2.imread(my_image,-1)

image[6][6] = 0

print(image)
```

```
[[  0   0   0   0   0   0   0   0]
 [  0 255 255 255 255 255 255   0]
 [  0 255   0   0   0   0 255   0]
 [  0 255 255 255 255 255 255   0]
 [  0 255 255   0   0   0   0   0]
 [  0 255 255   0   0   0   0   0]
 [  0 255 255   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0]]
```



Lists

Indexing and slicing a list:

Remember that indexes start from 0!

```
list1[x]        #starts indexing from the beginning of the list
list1[-x]       #starts indexing from the end of the list
list1[x:y]      #slices the list and returns the part from x to y
list1[:]        #copies the list
```

Some methods

```
list1.pop(idx)                    #removes the element at index idx
list1.remove("something")         #removes the first occurence of "something"
list1.append("something")         #adds something at the end of the list
list1.len()                       #returns the length of the list
list1.insert(idx, "something")    #adds something at the index idx
list1.extend(list2)               #adds the elements of list2 to the end of list1
list1.copy() | same as list1[:]   #returns a copy of the list
```

When in doubt, always check the documentation!

Exercise

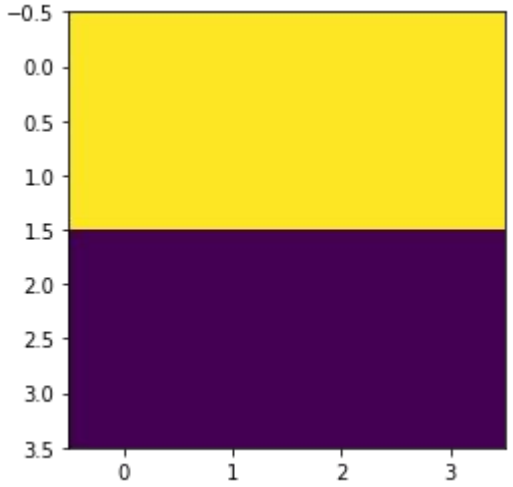
1. Declare a list that contains only 4 empty lists
2. Using the loop structure for, fill the first two lists with 4 elements each that are an integer different from 0 (*e.g. 5*) and fill the last two lists with 4 elements each that are equal to 0.

Exercise

```
nested_lists = [[],
                []],
                []],
                []]

for i in range(4):
    for j in range(4):
        if i < 2:
            nested_lists[i].append(5)
        else:
            nested_lists[i].append(0)

pyplot.imshow(nested_lists)
pyplot.show()
```




```
[[5, 5, 5, 5], [5, 5, 5, 5], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Sets

Sets

A brief word on sets.

Sets are unordered and cannot contain two identical values.
Different from tuples and lists, to declare a set you must type:

name_of_my_set = set()

If you try *name_of_my_set = {}* you will be declaring a Python dictionary, which is another completely different structure, which will be covered in the next lesson.

Sets

Sets cannot contain lists. However, it still accepts any of the data types we spoke before: numeric, textual and boolean.

Exercise

Define a new set and using the nested list defined in the previous exercise, run through the list adding every element to the set. Then, print every element of the set individually.

Exercise

Define a new set and using the nested list defined in the previous exercise, run through the list adding every element to the set. Then, print every element of the set individually.

```
for i in range(len(nested_lists)):
    for j in range(len(nested_lists[i])):
        my_set.add(nested_lists[i][j])

print(my_set)
for item in my_set:
    print(item)
```

```
{0, 5}
0
5
```